(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 101 172 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
06.08.2003 Bulletin 2003/32

(21) Application number: **99912573.5**

(22) Date of filing: **16.03.1999**

(51) Int Cl.7: **G06F 17/30**

(86) International application number:
PCT/US99/05716

(87) International publication number:
WO 99/048029 (23.09.1999 Gazette 1999/38)

(54) PERSPECTIVE TRANSFORMATIONS ON RELATIONAL DATABASE TABLES

TRANSFORMATION DER PERSPEKTIVE AUF TABELLEN VON RELATIONALEN DATENBANKEN

TRANSFORMATIONS DE PERSPECTIVES SUR TABLES DE BASES DE DONNEES RELATIONNELLES

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priority: **16.03.1998 US 39728**

(43) Date of publication of application:
23.05.2001 Bulletin 2001/21

(73) Proprietor: **MICROSOFT CORPORATION**
Redmond, Washington 98052-6399 (US)

(72) Inventors:
- **GRAEFE, Goetz**
  Bellevue, WA 98008-5634 (US)
- **ALGER, Jeff**
  Redmond, WA 98053 (US)

(74) Representative:
Beresford, Keith Denis Lewis et al
BERESFORD & Co.
2-5 Warwick Court,
High Holborn
London WC1R 5DH (GB)

(56) References cited:
WO-A-95/11487

- **J. C. NOSSITER: "Using Excel 5 for Windows" 1995 , QUE CORP. , USA XP002109365 page 256, line 1 - page 258, line 1 page 276, line 15 - page 281, line 3**
- **FLOHR, UDO: "Olap by Web", BYTE, USA, September 1997, pages 81 to 84**
- **C. B. DARLING: "Think Outside the OLAP Box", DATAMATION, USA, 15. April 1996, vol. 42, no. 8, pages 88 to 92**
- **B. DESMARAIS: "Using the Microsoft Excel Pivot Table for Reliability Applications", IEEE 34TH ANNUAL SPRING RELIABILITY SYMPOSIUM, USA, 18. April 1996, 13 pages**
- **E. LINDHOLM & S. MAEL: "DATAMATION's Feature Summary: OLAP Servers.", DATAMATION, USA, 01. May 1995, vol. 41, no. 8, pages 70 to 71**
- **M. FRANK: "BrioQuery 3.5", DBMS ONLINE, USA, February 1996, vol. , no. , pages 1 to 4**
- **M. GYSSENS ET. AL.: "Tables as a Paradigm for Querying and Restructuring", PROCEEDINGS 1996 ACM SIGMOD INTL. CONF. ON MANAGEMENT OF DATA, MONTREAL CA, 03. June 1996-06. June 1996, pages 93 to 103**
- **The OLAP Council: "OLAP and OLAP Server Definitions" ((c) 1995), accessible on-line at http://www.datamation.com/dataw/04bevalgls.html**

Printed by Jouve, 75001 PARIS (FR)

Description

Background of the Invention

[0001] The present invention relates to electronic data processing, and more specifically concerns new query operations for the manipulation of tables in relational databases .

[0002] A database is a collection of data in an organized structure. A typical database is stored in a computer as a set of records each having a number of fields for holding data items of a particular kind, such as character strings, numbers, or pointers to data located somewhere else. A relational database comprises any number of rectangular tables. Each table has a set of records; each record is referred to as a row of its table. Each record in the same table has the same number of fields. (However, some fields in a record may hold no data, indicated by a NULL value.) The fields of a table form a set of columns, which may have designated names that are not part of the data itself. The records do not have external to identify them individually. Instead, they are accessed by a key consisting of the contents of some combination of the fields; that is, a relational database may be considered to be a software-implemented content-addressable memory.

[0003] A database management system (DBMS, or database system) is computer software for storing, maintaining, and searching the data in a database. A DBMS usually includes facilities for increasing performance, reliability, and integrity, such as indexes, logging, and record locking. It always includes one or more interfaces for finding particular data from the database and for presenting these queries to a search engine. The engine searches the database and returns to the user a result, usually in the form of a relational table, which matches the specifications of the query.

[0004] The most widespread interface for relational databases is Structured Query Language (SQL). Although many variants of this interface language exist, standard versions have been defined by the American National Standards Institute (ANSI) and the International Standards Organization (ISO). Most present commercial realizations of SQL follow these standard versions, although many of them include language constructs in addition to those defined in the standard, or at different levels of compliance.

[0005] Relational databases and relational query languages treat data as a set of rectangular tables. Yet many databases are conceptually multidimensional, based upon axes such as time {day, month, year}, locale {store, city, state}, category {product, product_group}, actor {clerk, department, division}, payment {cash, check, credit}, and so forth. A user often finds it useful to think of such data as a collection of collections, and may wish to view them from different perspectives. In the above example, one perspective is a collection of records, where each record represents a locale, and contains a collection of monthly sales data for that locale; another perspective sees a collection of records (i.e., rows of a table) where each denotes a particular point in time, and the fields of each record (i.e., the columns of the table) collect sales figures for the different categories.

[0006] From this point of view, the ability to transform a database table from one perspective to another—to rotate the dimensions of the data—would be a valuable addition to the conventional capabilities of a query language such as SQL. In this context, to rotate perspectives or dimensions means to interchange a dimension represented in a table as a set of columns with a dimension represented as a set of rows. Conventional relational DBMS products and standards offer no direct operation for rotating perspectives. Although it is possible to formulate SQL queries to achieve this effect indirectly, such queries are large, complex, error-prone, slow, and hard to optimize into efficient execution plans, even when parallel processing is available.

[0007] Some conventional spreadsheet software allows a user to interchange data in a user-selected rectangle of cells to be interchanged in the same way that a matrix-algebra "transpose" operation relocates a matrix element a_{ij} to a_{ji}. In the Microsoft Excel product's Pivot Table feature, for example, a user selects a rectangle of cells, copies it into a temporary clipboard, points to a destination cell, and performs a "paste special" operation after selecting "transpose" from an options menu. With a suite of compatible application programs such as Microsoft Office, a user may even select data from a database table in the Microsoft Access database component, transfer it as a single object to the Excel component as a rectangle of spreadsheet cells, transpose the cells, then transfer the cells back into the Access database as a collection of records in the transposed format. Pivot operations in spreadsheets are described in J. C. Nossiter, USING EXCEL 5 FOR WINDOWS Que Corp, 1995) and in B. Desmarais, "Using the Microsoft Excel Pivot Table for Reliability Applications", IEEE 34TH ANNUAL SPRING RELIABILITY SYMPOSIUM (18 April 1996), pages 79-81.

[0008] Transposing data items in this manner is both clumsy and functionally limited. Even for small databases, the invocation of another application program merely to carry out a single query is wasteful. For large databases, the conventional requirement that transposed data reside in memory renders this method impossible. For client/server architectures using host-based search engines, there is no way to connect to a spreadsheet program for performing the operation. In any environment, transposition via spreadsheet requires manual intervention, and thus does not permit a transposition to form an internal part of a query within a database program. Such external operations cannot participate in the sophisticated reformulation, rewriting, and other optimization procedures of conventional database-query processors and other search engines. On a more conceptual level, fundamental differences between spreadsheets and relational database tables prohibit the desired types of transposition. For example, the names of the columns or fields in a database table are not a part of the table itself; they do not form a record of the table in the way that column headings in a spreadsheet are a row of cells within the spreadsheet. Transposing a rectangle of cells in a spreadsheet thus cannot transform a column of cells into the names of columns when the spreadsheet rows return to the database progrumas records in a table.

[0009] Some non-relational database systems have operations similar to spreadsheet pivot operations. OLAP (on-line analytical processing) can perform a rotate operation upon multidimensional "cube" data, as noted in U. Flohr, "OLAP by Web", BYTE, September 1997, pages 81-84, in E. Lindholm et al., "DATAMATION's Feature Summary: OLAP Servers", DATAMATION, May 1985, pages 70-71, in M. Frank, "BrioQuery 3.5", DBMS ONLINE, February 1996, in "OLAP and OLAP Server Definitions", (OLAP Council, 1995), and in C. B. Darling, "Think Outside the OLAP Box", DATAMATION, 15 April 1996, pages 88-92.

[0010] The execution engine of the Microsoft SQL Server product has a strictly internal operation for splitting each item of a table update having the form (row_identifier, old_values, new_values) within a stream of update items into a "delete item" and an "insert item" which interchanges certain row and column values, and a similar operation for collapsing a "delete item" and an "insert item" into an "update item". These operations are not available to users and cannot participate in user queries. That *is,* the query processor uses them internally only for facilitating the efficient execution of certain functions performed while updating databases.

[0011] Thus, the database art could be significantly expanded by providing a facility for fast, efficient rotation of perspectives, especially for relational databases. Moreover, there is a need for rotation or transposition operations whose semantics and syntax integrate well into query languages such as SQL as natural extensions, and which can be optimized and executed in conventionally organized database query processors and other search engines without adding complex or idiosyncratic facilities.

Summary of the Invention

[0012] The present invention as defined in claims 1, 11 and 17 provides a "pivot" operation for transforming the rows (records) and columns (fields) of a table, as that term is defined in a relational database, so as to provide different perspectives into the data items in the table. The operation accepts an input table and a pivot specification, and produces an output table. It takes place in the interface-language organization in such a way that it can be easily integrated into conventional database query processors, search engines, and servers. The operation places data in the fields of specified table records into the same field of different records, using the values of one or more designated table column as the names of the fields themselves. Data in any further columns are grouped by data values in a pivoted table.

[0013] It is sometimes easier to perform other relational operations upon a database table from another perspective, even when the ultimate result will have the original perspective. Therefore, the invention as defined in claims 7, 15 and 19 also provides an "unpivot" operation as an inverse to the pivot facility. Also, sometimes it is desirable to unpivot a stored table or intermediate result.

[0014] These operations, along with a simple and intuitive way of incorporating them into database queries, simplifies the writing of queries and makes them less error-prone. For example, they reduce or eliminate the need for joining tables to themselves. The method of invoking the operations permits deep nesting of multiple operations with a simple and powerful syntax extension and well-defined semantics, and applies a familiar programming-language paradigm. Permitting text as method arguments in queries enhances the power and ease of use of the extended SQL language. Moreover, expanding the set of relational-algebra expressions available in this manner to nonprocedural query expressions may also be applied to other operations, such as sample, top, and rank.

[0015] Pivot and unpivot operations according to the invention are inherently compatible with many types of data-manipulation software, and system architectures, especially including relational databases. These operations can be integrated into such systems both at the language level (e.g., by means of intuitive extensions to SQL and other query languages) and at the processing level (e.g., query optimization and execution).

[0016] Integrating data from multiple databases into a single data-warehouse database frequently faces an "impedance mismatch" when the multiple data sources have mutually differing shapes or row/column ratios. Almost by definition, such databases can be extremely large. Normalizing such data may depend upon context: storing data in pivoted form or perspective may be optimal—or even necessary-for one schema, while another schema may prefer or require the unpivoted form. Therefore, adding pivot and unpivot operations can greatly benefit the combination of data from different sources, especially large amounts of data.

[0017] The new operations provided by the invention also expedite lower-level DBMS processing, even with limited system resources. The extensible syntax and clear semantics of the new operations facilitates automatic generation and optimization of complex queries, especially in the rewriting of queries for more efficient execution. Even purely internal DBMS functions, such as update processing for index and integrity maintenance and other purposes, can benefit. The processing of SQL queries involving IN, OR, and UNION queries can be enhanced. Many optimization techniques already employed for GROUP BY queries are routinely adaptable to processing pivot and unpivot queries. Conventional execution algorithms including parallel-processing techniques for these queries apply to pivoting tables or query results, including unsorted and partitioned tables and results.

[0018] Other features and advantages of the invention, as well as variations within the scope of the invention, will appear to those skilled in the art from the following detailed description.

Brief Description of the Drawing

[0019]

Fig. 1 is a block diagram of a computer network environment for the invention.
Fig. 2 is a diagram of a database management system for hosting the invention.
Fig. 3 is a flowchart of the functions performed by the DBMS of Fig. 2.
Fig. 4 illustrates examples of pivot and unpivot operations according to the invention.
Fig. 5 is a flowchart of a pivot operation according to the invention.
Fig. 6 is a flowchart of an unpivot operation.

Detailed Description

Exemplary Operating Environment

[0020] Datanase management systems are implemented in many different types of data-processing systems, including standalone personal computers, midrange and mainframe computers, peer-to-peer and client/server networks, and wide-area distributed systems of many architectures. All data-processing systems are suitable environments for the present invention. For purposes of exposition, however, the invention will be described in connection with a conventional client/server computer system 100, shown in Fig. 1. Network wiring 110 interconnects a number of personal computers (PCs) 120 to a server 130 via network adapters 121 and 131. Server 130 includes a storage subsystem 132 for holding the large amounts of data in typical enterprise databases. Other system architectures are also suitable environments for the invention; for example, units 120 may be terminals connected to a mainframe or midrange computer 130, or unit 130 may itself comprise a PC coupled to PCs 120 in a peer-to-peer network. For small and modest databases, the entire system 100 may comprise a single PC acting as both client and server. Likewise, file storage may be distributed among a number of different machines. Fig. 1 shows schematic representations of an external storage medium 133 which may store client and server software for distribution and downloading to clients, and another medium 134, such as a diskette, for offline storage of database tables.

[0021] Figure 1A and the following discussion are intended to provide a brief, general description of a personal computer 120. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by a personal computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computes, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

[0022] Fig. 2 is a block diagram of a typical conventional client/server database management system 200 capable of operating in system 100, Fig. 2. A client application program 210 executes within each PC 120, under a PC operating system 220 such as Microsoft Windows95. Among other functions, client application 210 contains a facility 211 for accepting database queries from a user at a PC 120. In addition to user entries, other application programs 230 executing in some of the PCs 120 may present queries to DBMS client 210, via predefined host-language application-program interfaces (APIs) 231.

[0023] Within server 130, a DBMS server application 240, such as Microsoft SQL Server, executes under a server operating system 250 such as Microsoft Windows NT. DBMS program 240 provides services for creating, querying, maintaining, and modifying a number of relational databases, exemplified by database 260. Program 240 may employ the file-system services 251 of operating system 250, or may provide its own file system. Operating system 250 could execute a separate instance of the entire DBMS application for each request from a client 210. For greater efficiency, however, program 240 gives each client connection a separate thread 242 in the DBMS kernel. Further, this thread may be a native operating-system thread, which carries with it all the Windows NT mechanisms for process memory protection, better access to storage devices, and so forth. Search engine 241 processes queries and other requests from individual clients 210 upon tables 261 of a database 260, as described more fully below. It also enforces database integrity with conventional facilities for record locking, atomic transactions, etc. In Microsoft SQL Server, the interface language between query facility 211 and search engine 241 is Transact-SQL, which provides most of the functions of the standard ANSI SQL 89 and ANSI SQL 92 languages, plus extensions for providing greater flexibility and programmability.

[0024] Fig. 3 illustrates some conventional functions 300 of search engine 241, Fig. 2, for processing a query transmitted from any of the client applications 210. SQL is a nonprocedural language because an SQL query is a specification of properties or predicates of a desired result, rather than a sequence of steps for obtaining the result. That is, a query such as SELECT year, quarter, sales FROM Narrow WHERE sales < (SELECT AVG (sales) FROM Narrow) ORDER BY year, quarter specifies the properties of an output table. The columns of the output table correspond to the columns named year, quarter, and sales taken from an input table named Narrow. The output-table rows (records) are to be ordered (i.e., sorted) by year and then by quarter within each year value. The records from the input table which appear in the output are only those where the value of sales is less than the average value of all values of sales in the table named Narrow. The nested subquery SELECT AVG(sales) FROM Narrow generates a table having only a single column and a single row containing the average value of sales in the Narrow table. The manner and sequence in which the records of the input table are accessed, and other details of the procedure or plan for building the output table are not defined by the query itself.

[0025] When search engine 241 receives a query, it parses the query into an internal or tokenized form, as shown in step 310. Validation step 320 ensures that the data named in the query actually exists in the database, and checks data and integrity constraints. It may expand some parts of the query, such as macros and views, at 321. Output 322 reports any errors back to the user or other source of the query. All but the most limited search engines perform extensive optimization upon the query, as indicated in step 330. Optimization may involve rewriting the query by combining or splitting portions of the query, rearranging operations and subqueries, etc., and other methods, such as sequencing accesses to the records of stored database tables, and modifying functions For each candidate execution strategy, they calculate a cost value representing the computing time or resources required to execute the query using this strategy, and then select one strategy among all the possible candidates. Although the art of designing these optimizers is complex and arcane, those adept in it have the ability to adapt conventional optimizers so as to include new query functions of various types; designers of translators for other, more procedural languages also routinely construct optimizers of this same general class. A survey paper, M. Jarke and J. Koch, "Query Optimization in Database Systems," *ACM Computing Surveys* 16, 2 (June 1984), p. 111, discusses the construction of database query optimizers in more detail.

[0026] The output of step 330 is a query evaluation plan (or simply "plan") for answering the query. Step 340 compiles this plan into a procedural form, usually represented as a conventional function tree. Step 350 may then run a simple tree-traversal algorithm for executing the plan against the database objects. The output of step 350 is the result of the query, in the form of an output table returned to the source of the query. Search engines other than the one described herein may combine or divide the individual steps 300, or may omit or add steps. Another survey paper, G. Graefe, "Query Evaluation Techniques for Large Databases," *ACM Computing Surveys* 25, 2 (June 1993), p. 73, hereby incorporated by reference, addresses the subject of query execution, and cites a number of references providing additional description and discussion. Again, the steps of the newest search engines are specifically designed for easy extensibility to accommodate new syntax, new query function, optimization knowledge, and execution technology.

Pivot and Unpivot Operations

[0027] Fig. 4 shows the structure of a pivot operation according to the invention. This operation fits into the hierarchy of SQL operations at the level of relational algebra. Those who design relational database systems and interfaces divide query processing into three levels. Because the mathematical theory of relations provides the conceptual framework for this type of databases, the first and second levels are frequently called the relational calculus and the relational algebra.

[0028] The relational calculus, like any calculus, deals with a high-level description or specification of a desired result, without naming any operations, procedures, or other method for obtaining the result. That is, it merely expresses the definition of a desired result relation (table) in terms of existing relations in a database. The query SELECT employee.name, department.name FROM employee, department WHERE employee.dept_id = department.dept_id, for example, describes the properties and constraints of an output table in terms of one or more input tables in terms of a typical member of the result relation and a qualification representing the defining property of the result members. The relational calculus provides the foundation for a formal, exact understanding of databases, tables ("relations"), and queries, and has found a commercial realization in the query components of the database language SQL, now an ANSI/ISO standard. Given the important role SQL plays in data-management products, extending database functionality for the real world requires that any added functionality should become a syntactically and semantically clean extension of the SQL language.

[0029] Relational algebra is more operationally oriented (yet equivalent to) relational calculus. Operations or functions in relational algebra consume one or more input tables and produce an output table according to a rule. For example, the relational operation JOIN [employee.dept_id = department.dept_id] (employee, department) combines the tables named employee and department along a common column or field named dept_id in both tables. (This is analogous to an operation such as addition, which consumes two numbers and produces a third, as, for example, the operation "4+5" produces "9".) Key characteristics of relational algebra are that: (1) operations consume and produce objects of the same type, namely relations; (2) operations can be nested into arbitrarily complex structures; and (3) new operations can be added. In the relational algebra, input objects not only have inputs, but may also carry tags denoting additional information. In the example immediately above, the join operation not only specifies the two relational-algebra expressions, namely the two tables (employee, department) to be joined, but also names a "join predicate" specifying how they are to be joined: along equal values of a particular column in each table, [employee. dept_id = department.dept_id].

[0030] Some very useful query operations are difficult to express at the relational-calculus level, yet they integrate easily and cleanly into the relational-algebra level. For example, OUTER JOIN, a variant of the relational JOIN operation, does not fit easily or cleanly into the simple SELECT... FROM... WHERE query syntax. Therefore, ANSI/ISO permit a limited set of relational-algebra expressions in place of tables in the from clause, e.g., SELECT employee.name department. name FROM employee LEFT OUTER JOIN department ON employee.dept_id = department.dept_id. That is, there is a precedent for extending a relational-calculus query with a relational-algebra expression, although such extensions have been thus far restricted to variations of JOIN operations. Relational-algebra operations often participate in the optimization of queries having selections, projections, aggregations, and other nonprocedural specifications at the relational-calculus level, as in block 330, Fig. 3.

[0031] Query-execution plans constitute the third and lowest level of query processing. Although the nesting of relational-algebra operations may indicate an order of execution, algorithms or sets of particular instructions for producing intermediate results occur at the level of execution plans, rather than at the higher levels. For example, there are three basic methods for performing relational JOIN operations: nested loops, merge-join, and hash-join; and each method has a large number of variants. Execution plans clearly indicate the choice among such alternatives, and are formulated at the lowest level of query processing in block 340, Fig. 3.

[0032] Because relational query processing is defined very precisely and within a definite structural framework, it is important to define any new functionality at all three levels: language extensions, relational-algebra operations, and execution plans. The invention may provide the pivot and unpivot functions as new relational-algebra operations that participate explicitly in SQL queries, as extensions to the language.

[0033] The formal definition of a pivot operation, for an input table-expression in first normal form that is a valid query expression, is: Table. PIVOT (<value_column> FOR <pivot_column> IN (<pivot_list>)); the output pivoted table then is also a valid first-normal-form table. The text between the outermost parentheses constitutes the specification of the pivot operation. The first two columns in the pivot specification must be columns in the pivot operation's input table. These columns will not appear in the pivot operation's output table. Instead, each value in the pivot list within the pivot specification defines a new column in the pivot operation's output table. In the input table, elements in the pivot list appear as values in the pivot column. Corresponding values in the value column become values in the new columns in the output table. All columns of the input table not included in the pivot specification, called "grouping columns," are carried over to the output table.

[0034] In the example 400 set forth in Fig. 4, pivoting input table 410 in accordance with specification 420 produces output table 430. Pivot column 411, named Quarter in the input table 410 named Narrow, becomes four columns 431, 432, 433, and 434 in the output table 430. The names of these columns are the four distinct values, Spring, Summer, Fall, Winter, that appear as values in column 411, and that also appear in the value list following the keyword IN in 420. The sales numbers in value column 412 appear as values in corresponding ones of the four columns 431-434, but pivoted or rotated, so that sales figures for the same region and year are in the same row. Grouping columns 413-414 appear as columns 435-436 in output 430. In the output, the rows are grouped by equal values of the first grouping column 413, and then by equal values of the second grouping column 414, just as though specification 420 had contained an SQL clause of the form GROUP BY Region, Year. In this example, the effect of the pivot operation is to modify the perspective along which the data is viewed. Input table 410 presents data trends primarily by year for the Narrow regions of a company, whereas output table 430 allows seasonal tracking by quarters. (It should be recalled here that the rows in an relational table do not have names, and have no particular order. Columns do have names and are sorted - that is, they are presented in the order their names appear in a query.) The pivot operation converts an input table having relatively many rows and relatively few columns into a result table having fewer rows and more columns.

[0035] The pivoted columns in the output table have the same data type (numeric, varchar, etc.) as the data in the value column of the input table. The value column, pivot column, and pivoted columns comprise simple data, rather than computed expressions. The order of the columns in both tables is not significant, as in ANSI SQL; columns can be referenced only by name, not by position. Although table 430 is shown sorted by values of grouping columns Region and Year, the pivot operation does not imply any particular sorting or ordering of the rows.

[0036] As mentioned above, a row in the input table does not appear in the output if its value does not appear in the pivot list. The input-table rows are grouped by equal values of any grouping columns, with respect to the definition of equality. Within each group, each row of the input table has a mutually distinct value in the pivot column. Each group results in one output row. For output columns not having a corresponding input row, the value is NULL, a special value defined in SQL.

[0037] Fig. 5 is a flowchart 500 of the steps carried out by modules 300, Fig. 3, of search engine 241 in Fig. 2 for a pivot operation. Block 510 receives a query from a user at a client terminal 120, Fig. 1, or from some other source as described herein. Step 520 identifies or selects which table 261 in database 260 is to serve as the input table of the operation. Step 521 identifies which column of the input table is to be the pivot column, step 522 identifies from the pivot list which pivot-column values participate in the pivot, and step 523 selects which column of the input is the value column. Step 530 constructs the output as another table 261. Step 531 emplaces a separate pivoted column for each data-item value in the pivot list. Step 532 constructs the grouping columns, if any. (As mentioned previously, these are any additional columns of the input table not identified in the pivot specification.) Step 540 inserts the data-item values of the value column into the rows of the output table as described previously; one method is by transposition, as indicated by step 541. Another way to express this transposition is that each data item in the value column is placed into one of the pivoted columns, namely, that column whose name is the same as the data value in the pivot column of the input table. Step 550 groups the output-table rows by equal values of any grouping columns. Finally, step 560 stores the output table in database 260, Fig. 2.

[0038] The unpivot operation is the inverse of the pivot operation, and is formally defined as (table_expression| query_expression>.UNPIVOT (<value_column> FOR <pivot_column> IN (<column_list>). The meanings of the terms are the same as for the pivot operation. Applying a pivot and an unpivot operation having the same specification to an input table restores the input table to its original state. In the example shown in Fig. 4, applying the unpivot operation 440 to pivoted table 430 restores table 410; the two named columns Sales 412 and Quarter 411 replace columns 431-434.

[0039] Other database-system operations can be considered as inverses of each other, such as the grouping/splitting and merging/collapsing pairs mentioned in M. Gyssens, et al., "Tables as a Paradigm for Querying and Restructuring", PROCEEDINGS 1996 ACM SIGMOD INTL. CONF. ON MANAGEMENT OF DATA, Montréal, Qué., Canada, 3-6 June 1996, pages 93-103. Pivot/unpivot are also inverses of each other.

[0040] For each row in an input table, the unpivot operation generally produces one row of an output table per pivoted column. (However, a null value in a pivoted column does not generate an output row.) All columns in the pivot list must have the same data type in the input, and the entries in the value column of the output will have this type. Unpivoting a table increases the number of its rows and decreases the number of columns. Again, although Fig. 4 shows table 410 sorted by grouping-column values, the unpivot operation implies no row sorting in the output.

[0041] Fig. 6 is a flowchart 600 of the steps carried out by modules 300, Fig. 3, of search engine 241, Fig 2 for an unpivot operation. Step 610 receives the unpivot operation and its specification. Because unpivot is defined to be the inverse of pivot and to restore a pivoted table exactly to its unpivoted form, the specification of an unpivot is not complementary to that for a pivot, but rather has exactly the same form as that of the specification which created the pivoted table in the first place, as shown at 440 in Fig. 4. Again, step 610 may receive the operation from a user query or any other source. Step 620 identifies or selects the pivoted table such as 261 to be unpivoted. This table need not have been actually pivoted in a previous operation, but it normally will have been; that is, the pivot operation is generally used to achieve any initial rotation of perspectives, and unpivot is generally only used to restore a table to an original unpivoted form in a clean, simple manner. Step 621 uses the specification's pivot list to identify which columns are the pivoted columns to be rotated or transposed. Steps 622 and 623 identify the names of the value column and the pivot column. These names are included in the specification because they do not appear anywhere within the pivoted table (at least, unless a previous pivot had kept a side table retaining this information).

[0042] Step 630 constructs the pivot table which is to be produced by the unpivot operation. Steps 631 and 632 form the pivot and value columns in the pivot table, using the names supplied in steps 622 and 623. Step 633 builds grouping columns in the pivot table, one for each of the pivoted-table columns not included in the unpivot specification received in step 610. Step 640 transposes the data items from the pivoted table into the unpivoted table constructed in the preceding steps. Names of the pivoted columns become data items in different rows, and the data items in the pivoted column go into the new value column, in the rows having respectively the same pivot-column values as the name of the pivoted column that they were in, in the original (pivoted) table. Step 650 groups rows by equal values of the grouping-column rows. Step 660 stores the table away in database 260, Fig. 2.

[0043] Correlation variables are not permitted within the specification of pivot and unpivot operations, because the pivot, value, and pivoted columns are simple data, and not computed values. These new operations have no bearing upon whether or which correlation variables are permissible in a query expression to which the operations are applied. If ANSI SQL permits defining table and column aliases for a query expression in which a pivot or unpivot operation does not occur, then it is acceptable to define such aliases for the query expression including a pivot/unpivot operation, but not for the query expression excluding the operation. For example, if Table1 AS Table2 (col1, col2) is permissible, then Table1.PIVOT (...) AS Table2 (col1, col2, ...) is acceptable, but Table1 AS Table2 (col1, col2).PIVOT (...) is not.

[0044] The pivot and unpivot operations may employ any number of conventional optimization techniques in block 330, Fig. 3. Because these operations are part of the relational-algebra level, an algebraic query optimizer may be the most appropriate vehicle for realizing optimization techniques. Other optimization frameworks may be applicable as well.

[0045] Some additional optimization techniques may employ specific properties of the new operations. Obviously, a neighboring pair of pivot and unpivot operations may cancel each other, and may then be removed from a query. An optimizer should recognize that the grouping columns in the pivoted output table functionally determine the pivot and value columns, and therefore form a relational key of the result table. (This is very similar to the grouping columns in a conventional GROUP BY operation.) In an unpivot output table, the grouping columns together with the pivot column functionally determine the value column. These properties can assist in estimating the number of output rows for selectivity estimation and query-cost calculation for comparing alternative execution plans. They also may find utility in generating conditions for applying rewrite rules in simplifying the execution of a query. If a table is vertically partitioned, an operation to reassemble complete rows and a subsequent unpivot may cancel each other, eliminating both operations.

[0046] A conceptual similarity of the pivot operation to an SQL GROUP BY clause allows many techniques and rules for optimizing queries having that clause to serve as well for the new operations. Typical examples include: (1) pulling a pivot above a join, so as to reduce the grouping input's size, or to enable more efficient join algorithms; (2) pushing a pivot below a join to reduce the join input or to employ more efficient execution plans for the pivot; (3) merging two adjacent pivots, possibly effectively eliminating one of them; and (4) splitting a pivot into two parts, then pushing one of the parts through a join or across a process boundary, as a local/global aggregation in a parallel execution environment. In general, a query predicate on the grouping columns and a projection operation--including expressions that compute additional columns--can be moved through (either above or below) a pivot/unpivot operation in the same manner as a grouping operation.

[0047] Certain query predicates are more efficient to implement--and also easier to express--when treated as predicates (i.e., qualifications) against a pivot result table. For example, comparing two pivoted columns to each other is straightforward to express and efficient to implement, whereas the same predicate applied to the pivot input table requires complex, inefficient nested queries. Therefore, rewriting the query to include a pivot/select/unpivot sequence of operations can be used to optimize such queries. For example, consider a query to select table rows in which Sales in the Fall exceeds Sales in the Spring in table 410, Fig. 4. Pivoted table 430 can accommodate this query as a comparison between the Spring and Fall columns 431 and 433, whereas the original table 410 requires joining the table to itself in order to perform the comparison. Using the tables of Fig. 4, such a larger query including pivot and unpivot operations (in conventional multiple-line format) might be:

```
(SELECT * FROM
Narrow.PIVOT (Sales FOR Quarter IN (Spring, Summer, Fall, Winter))
WHERE Fall_Sales>Spring_Sales)
.UNPIVOT (Sales FOR Quarter IN (Spring, Summer,
Fall, Winter))
```

[0048] Execution plans useful for block 340, Fig. 3, can be derived by those skilled in the art from conventional plans for grouping operations. In particular, plans based upon looping, indexing, streams, sorting, and hashing come readily to mind. Early-aggregation sorting and hybrid hashing are useful variants. Pivot/unpivot operations are amenable to parallel-execution environments, including parallel algorithms such as shared-memory, distributed-memory, shared-disk, and cluster machines. Local/global aggregation has already been mentioned as a possibility.

[0049] Unpivot operations require only a single-input, single-output plan that produces multiple output records for each input record. This operation can be easily executed in parallel on shared-memory, distributed-memory, shared-disk, and cluster machines.

Variations and Extensions

[0050] A number of variants and extensions to the above embodiment may be useful for some applications, either alone or in combination with each other. The most obvious, of course, are the replacement or augmentation of notational conventions, such as the dot invocation separator, and the rearrangement of components.

[0051] The pivot/unpivot operations as thus far described place limitations upon column names in the pivoted or output table. Column names in standard SQL must be character strings without spaces. Because column values may have spaces, and pivot changes values into names, a method can easily be devised to employ quoted identifiers and literals as column names. Likewise, it would be simple to represent column values having data types other than character strings as printable and readable representations for column names. Conventional name manipulations, such as concatenations of names might be useful. For example, a pivot list might contain column aliases using a keyword such as SQL AS. (In Fig. 4, if the Quarter values were "1" through "4" instead of season names, the specification of 420 might read (Sales FOR Quarter IN (1 AS "Spring", 2 AS "Summer", 3 AS "Fall", 4 AS "Winter"). In addition, AS might be used to rename pivot-result columns in any context, and user-defined functions could be supplied for converting complex column names, or for conforming names to specific limitations of an SQL implementation.

[0052] Semantic extensions might include pivoting and unpivoting multiple columns in a single step. For example, operation 450 in Fig. 4 replaces the three columns 411, 412, and 414 with eight columns, instead of the four columns 431-434 of result table 430. That is, the set of pivot columns represent a Cartesian or outer product of all the pivot lists. A convention for naming the pivoted columns might merely involve concatenating the names, such as Sales_1996_Spring, etc. A multicolumn unpivot operation with the same specification could decode such names back into their original form, so as to provide a true inverse for this extension. A further extension would permit multicolumn pivoting in steps. For example, it might be desired to apply a further pivot to already pivoted table 430 about column 436, to produce the eight columns described above. Rather than unpivoting table 430 and then applying a multicolumn pivot, an extended form allows a list of columns in place of the value column, e.g., Wide.PIVOT ((Spring, Summer, Fall, Winter) FOR Year IN (1996, 1997)). Optimizer 340 and compiler 350 can easily collapse these uperations into a single plan for execution.

[0053] The pivot operation (but not unpivot) can support conventional SQL aggregation or grouping functions such as MIN, SUM, AVG, and even COUNT, for the value column at step 540. In that case, the limitation to a single row per group can be lifted. Of course, the type of the new column might differ from that of the original. The following example, based upon Fig. 4, illustrates a query using aggregation:

(SELECT Year, Quarter, Sales FROM Narrow)
.PIVOT (SUM(Sales) FOR Quarter IN (Spring, Summer, Fall, Winter)

This query consolidates Sales for the East and west regions into a single sum representing the entire company for each Year. In versions where grouping functions are allowed, the implementation could specify the implicit application of a particular function, such as SUM, in all cases where a pivot operation would otherwise produce duplicate primary keys in different rows of the pivoted table. Pivots with grouping cannot be reversed, because the aggregation loses information detail; grouped output in standard SQL cannot be reversed for the same reason. Although this extension prevents unpivot from functioning as a true inverse, an embodiment preserves this capability by adding an internal "side table" that saves all the original values.

[0054] Another powerful extension adds a capability for replacing the list of literal column names in a pivot or unpivot operation with a SELECT query. More complex processing would involve running the auxiliary query first, then binding the list of pivoted columns using the result of the auxiliary query; that is, the auxiliary query requires interleaved compilation and execution in blocks 340 and 350, Fig. 3. The execution requires computing the query expression that is to be pivoted, as well as running a query against the result.

[0055] The pivot specification might omit the pivot list entirely, supplying a default query instead. For example, omitting the clause IN (Spring, Summer, Fall, Winter) from query 420 could substitute a default query SELECT DISTINCT Quarter FROM Narrow. Because this causes query 420 to reference the input table twice, it would be useful to introduce a dedicated name for an operation's input table, analogously to the name "this" in C^{++}. Operation 420 would then become:

Narrow.PIVOT (Sales FOR SELECT DISTINCT Quarter FROM INPUT).

[0056] Instead of requiring a pivot list, the unpivot operation might allow a specification of all but the pivoted columns in the operation's input. In the example 400 as modified above, the inverse operation could be specified as (see 440, Fig. 4:

Wide.UNPIVOT (Sales FOR Quarter IN (Spring, Summer, Fall, Winter)) or as (460, Fig. 4):

Wide.UNPIVOT (Sales OVER (Region, Year)).

Supporting OVER in this context necessitates determining the set of pivoted columns from the input table, and thus requires the ability to process auxiliary queries, as described above. The IN and OVER clauses can be combined, permitting one or more columns to be a pivoted column as well as a grouping column. A situation where this might make sense from the application's perspective is the inclusion of Spring sales in each output row, in order to allow computation of sales growth since the first quarter for each subsequent quarter.

[0057] In some tables, a set of columns can be more or less orthogonal or independent; for example, columns named "City" and "Month" are likely to have table entries for all cities for all months. Other column sets are hierarchical--such a "Locations" table having "State", "City", and "Store" columns--and their data is sparse; that is, very few cities will occur in multiple states, and few cities will have multiple stores. In the latter case, the use of two IN clauses leads to ungainly syntax and semantics in a pivot operation. However, employing a list of pivot columns instead of a single pivot column ameliorates this problem. ANSI SQL's concept of "row values" is appropriate to this case. Typically, although not always, it is more convenient to specify a query as the pivot list, rather than a list of literal column names. An exemplary form might be Locations.PIVOT (SalesVolume FOR (City, Store) IN (SELECT City, Store FROM Outlets)).

[0058] The pivot and unpivot operations could also find utility in the internal operation of query processors 300, Fig. 3 In addition, referential-integrity constraints need to be enforced only for deleted candidate keys and new foreign keys; using pivot/unpivot or a similar rotation could collapse deletion and insertion items pertaining to the same key value, and thus potentially eliminate some integrity checks as redundant. Moreover, in a conventional query having a very large IN clause, an internal unpivot operation invoked implicitly by the search engine can map a single very complex row containing many literals or parameters as columns into a set of rows that can be matched against database tables, using conventional join methods such as loops, index, merge, and hash join, and their parallel-processing variants. Similar internal invocations of pivot/unpivot operations may be useful in OR and UNION queries, which are often equivalent to IN clauses.

Claims

1. A method performed by a computer (500) of transforming data from an unpivoted input relational database table (410) into a pivoted output table (430),

   characterized in that all of the following steps are performed entirely within a relational database management system (200) for processing tables having rows and columns of items containing data-value items, and having items containing column names stored separately from and in a different format from that of the data-value items, without converting to and from a different overall table format, the steps performed by the computer comprising:

   selecting (521) the separately stored name of a first ofthe columns of the input table as a pivot column;
   selecting (523) the separately stored name of a second of the columns of the input table as a value column;
   converting (531) a set of the items stored as data values in the pivot column directly into items stored as column-name items in the output table denoting pivoted columns in the output table;
   placing (540) items stored as data values in the pivot column into respective data-value items in corresponding different ones of the pivoted columns of the output table;
   storing (560) the output table directly in the database management system, without converting it to or from a different format.

2. The method of claim 1 ***characterized in that*** the placing step comprises, for each data-value item in the value column located in the same row of the input table as a particular one of the set of data-value items in the pivot column, placing the particular one data-value item into a certain one of the pivoted columns of the pivoted output table the certain one pivoted column having a name corresponding to the particular one data-value item in the pivot column.

3. The method of claim 1 ***characterized in*** a further step of selecting (522) certain ones of the set of data-value items in the pivot column as a pivot list, and wherein the converting step converts only those data-value items in the pivot list into the column-name items.

4. The method of claim 3 ***characterized in* that** the converting step does not convert any rows in the input table having NULL data-value items in the pivot column.

5. The method of claim 1 ***characterized in that*** the input table has at least one column (413) other than the pivot and value columns.

6. The method of claim 5 ***characterized in*** grouping (532) the rows of the output table by equal values of the data-value items in the at least one other column.

7. A method (600) performed by a computer of transforming data from a pivoted input relational database table (430) into an unpivoted output table (410),

   characterized in that all of the following steps are performed entirely within a relational database management system (200) for processing tables having rows and columns of items containing data values, and having items containing column names stored separately from and in a different format from that of the data-value items, without converting to and from a different overall table format, the steps performed by the computer comprising:

   selecting (623) the separately stored name of a first of the columns of the input table as a pivot column;
   selecting (621)a plurality of the separately stored names of the columns of the input table as a pivot list;
   creating (632) in the output table a value column having a selected name stored in a column-name item and a plurality of separately stored data-value items;
   placing (640) items stored as column-name items in the pivot list into respective data-value items in corresponding different ones of the value column of the output table;
   storing (660) the output table directly in the database management system, without converting it to or from a different format.

8. The method of claim 7 ***characterized in that*** the placing step comprises, for each particular data-value item in each particular column of the input-table columns in the pivot list, placing the particular data-value item into the a data-value item of the value column of the unpivoted output table in a row which also contains a data-value item in the pivot column corresponding to the particular column of the input table.

9. The method of claim 7 ***characterized in that*** the input table has at least one column (435) other than the columns in the pivot list.

10. The method of claim 9 ***characterized in that*** the grouping step groups the rows of the output table by equal values of the data-value items in the at least one other column.

11. A relational database system (200) having a number of clients (210) and a search engine (240) including modules for parsing (310), optimizing (330), and executing (350) a query (420, 450) from one of the clients containing a pivoting operation specifying

a relational input table (410) having rows and columns of data values and having column names stored separately from the data values in the columns,
a name of a pivot column (411) in the input table, and
a name of a value column (412) in the input table,

characterized in that the search engine transposes (500) data values in the value column about the pivot column, converting at least some of the data values into separately stored and formatted column names so as to construct a pivoted output table directly from the input table without converting either table to or from a different format.

12. The system of claim 11 where the pivoting operation further specifies a pivot list (522) of data values in the pivot column,
characterized in that the data values in the pivot column are transposed based upon the data items in the pivot list.

13. The system of claim 12 where the input table includes columns (413) other than the pivot and value columns,
characterized in that the search engine groups (550) rows of the output table by equal values of the data-value items in the other columns.

14. The system of claim 12 where a query (440, 460) from one of the clients contains an unpivoting operation specifying

a pivoted relational input table (430) having rows and columns of data values and having column names stored separately from the data values in the columns,
a name (411) for a value column,
a name (412) for a pivot column, and
a pivot list including names of at least some of the columns (431-434) in the pivoted input table,

characterized in that the search engine transposes (600) the column names in the pivot list about the pivot column, converting those column names into separately stored and formatted data values so as to construct an unpivoted output table directly from the input table without converting either table to or from a different format.

15. A relational database system (200) having a number of clients (210) and a search engine (240) including modules for parsing 310), optimizing (330), and executing (350) a query (440, 460) from one of the clients containing an unpivoting operation specifying

a relational input table (430) having rows and columns of data values and having column names stored separately from the data values in the columns,
a name for a pivot column (412), and
a pivot list including names of at least some of the columns (431-434) in the input table,

characterized in that the search engine transposes (600) the column names in the pivot list about the pivot column, converting these names into separately stored and formatted data values so as to construct a pivoted output table directly from the input table without converting either table to or from a different format.

16. The system of claim 15 where the input table includes columns (435) other than the pivot and value columns,
characterized in that the search engine groups rows of the output table by equal values of the data-value items in the other columns.

17. A medium (133) carrying representations of instructions for causing a suitably programmed computer to perform a method (500) of transforming data from an unpivoted input relational database table (410) into a pivoted output table (430),
characterized in that all of the following steps are performed entirely within a relational da-

tabase management system (200) for processing tables having rows and columns of items containing data-value items, and having items containing column names stored separately from and in a different format from that of the data-value items, without converting to and from a different overall table format, the steps comprising:

selecting (521) the separately stored name of a first of the columns of the input table as a pivot column;
selecting (523) the separately stored name of a second of the columns of the input table as a value column;
converting (531) a set of the items stored as data values in the pivot column directly into items stored as column-name items in the output table denoting pivoted columns in the output table;
placing (540) items stored as data values in the pivot column into respective data-value items in corresponding different ones of the pivoted columns of the output table;
storing (560) the output table directly in the database management system, without converting it to or from a different format.

18. The medium of claim 17 further carrying representations of instructions for causing a suitably programmed computer to perform a method of transforming data from a pivoted input relational database table into an unpivoted output table in the same relational database system,

characterized in that all of the following steps are performed entirely within a relational database management system for processing tables having rows and columns of items containing data values, and having items containing column names stored separately from and in a different format from that of the data-value items, without converting to and from a different overall table format, the steps comprising:

selecting the separately stored name of a first of the columns of the input table as a pivot column;
selecting a plurality of the separately stored names of the columns of the input table as a pivot list;
creating in the output table a pivot column having a selected name stored in a column-name item and a plurality of separately stored data-value items;
converting the column-name items in the pivot list to data-value items in the pivot column;
creating in the output table a value column having a selected name stored in a column-name item and a plurality of separately stored data-value items;
placing items stored as column-name items in the pivot list into respective data-value items in corresponding different ones of the value column of the output table;
storing the output table directly in the database management system, without converting it to or from a different format.

19. A medium (133) carrying representations of instructions for causing a suitably programmed computer to perform a method (600) of transforming data from a pivoted input relational database table (430) into an unpivoted output table (410),

characterized in that all of the following steps are performed entirely within a relational database management system (200) for processing tables having rows and columns of items containing data values, and having items containing column names stored separately from and in a different format from that of the data-value items, without converting to and from a different overall table format, the steps comprising:

selecting (623) the separately stored name of a first of the columns of the input table as a pivot column;
selecting (621)a plurality of the separately stored names of the columns of the input table as a pivot list;
creating (632) in the output table a value column having a selected name stored in a column-name item and a plurality of separately stored data-value items;
placing (640) items stored as column-name items in the pivot list into respective data-value items in corresponding different ones of the value column of the output table;
storing (660) the output table directly in the database management system, without converting it to or from a different format.

Patentansprüche

1. Von einem Computer (500) durchgeführtes Verfahren zum Transformieren von Daten von einer unpivotisierten Eingangstabelle (410) einer relationalen Datenbank in eine pivotisierte Ausgangstabelle (430),

dadurch gekennzeichnet, daß alle folgenden Schritte vollständig innerhalb des relationalen Datenbankmanagementsystems (200) ausgeführt werden, um Tabellen zu verarbeiten, die Zeilen und Spalten von Einträgen mit Datenwert-Einträgen sowie getrennt von den und in einem anderen Format als die Datenwert-Einträge gespeicherte Spaltennamen aufweisen, ohne daß auf ein und von einem anderen Gesamttabellenformat konvertiert wird, mit folgenden vom Computer ausgeführten Schritten:

> Auswählen (521) des getrennt gespeicherten Namens einer ersten Spalte der Eingangstabelle als Pivot-Spalte,
> Wählen (523) des getrennt gespeicherten Namens einer zweiten Spalte der Eingangstabelle als Wertspalte,
> Umwandeln (531) eines Satzes von Einträgen, die als Datenwerte in der Pivot-Spalte gespeichert sind, direkt in Einträge, die als Spaltennamen-Einträge in der Ausgangstabelle gespeichert sind, die pivotisierte Spalten in der Ausgangstabelle benennen,
> Anordnen (540) von Einträgen, die als Datenwerte in der Pivot-Spalte gespeichert sind, in entsprechende Datenwert-Einträge in entsprechend unterschiedlichen pivotisierten Spalten der Ausgangstabelle, und
> Speichern (560) der Ausgangstabelle direkt in dem Datenbankmanagementsystem, ohne sie in ein anderes oder aus einem anderen Format zu konvertieren.

2. Verfahren nach Anspruch 1, **dadurch gekennzeichnet, daß** der Anordnungsschritt für jeden Datenwert-Eintrag in der Wertspalte, der in der gleichen Zeile der Eingangstabelle wie ein bestimmter aus dem Satz an Datenwert-Einträgen in der Pivot-Spalte angeordnet ist, ein Anordnen des bestimmten einen Datenwert-Eintrags in einer gewissen der pivotisierten Spalten der pivotisierten Ausgangstabelle umfaßt, wobei die gewisse pivotisierte Spalte einen Namen aufweist, der dem bestimmten einen Datenwert-Eintrag in der Pivot-Spalte entspricht.

3. Verfahren nach Anspruch 1, **gekennzeichnet durch** einen weiteren Schritt zum Auswählen (522) gewisser aus dem Satz an Datenwert-Einträgen in der Pivot-Spalte als Pivot-Liste, wobei der Umwandlungsschritt lediglich Datenwert-Einträge in der Pivot-Liste in die Spaltennamen-Einträge umwandelt.

4. Verfahren nach Anspruch 3, **dadurch gekennzeichnet, daß** der Umwandlungsschritt keine Zeilen in der Eingangstabelle mit NULL-Datenwert-Einträgen in die Pivot-Spalte umwandelt.

5. Verfahren nach Anspruch 1, **dadurch gekennzeichnet, daß** die Eingangstabelle mindestens eine andere Spalte (413) als die Pivot- und die Wertspalte aufweist.

6. Verfahren nach Anspruch 5, **gekennzeichnet durch** ein Gruppieren (532) der Zeilen der Ausgangstabellen im Hinblick auf gleiche Werte der Datenwert-Einträge in der genannten mindestens einen anderen Spalte.

7. Von einem Computer ausgeführtes Verfahren (600) zum Transformieren von Daten von einer pivotisierten Eingangstabelle (430) einer relationalen Datenbank in eine unpivotisierte Ausgangstabelle (410),

dadurch gekennzeichnet, daß alle folgenden Schritte direkt innerhalb eines relationalen Datenbankmanagementsystems (200) ausgeführt werden, um Tabellen zu verarbeiten, die Zeilen und Spalten von Einträgen mit Datenwerten sowie Einträge mit Spaltennamen, die getrennt von den und in einem anderen Format als die Datenwert-Einträge gespeichert sind, aufweisen, ohne daß eine Konvertierung in ein anderes oder aus einem anderen Gesamttabellenformat durchgeführt wird, mit den folgenden vom Computer ausgeführten Schritten:

> Auswählen (623) des getrennt gespeicherten Namens einer ersten der Spalten der Eingangstabelle als Pivot-Spalte,
> Auswählen (621) mehrerer der getrennt gespeicherten Namen der Spalten der Eingangstabelle als Pivot-Liste,
> Erzeugen (632) einer Wertspalte in der Ausgangstabelle mit einem ausgewählten Namen, der in einem Spaltennamen-Eintrag gespeichert ist, und mit mehreren getrennt gespeicherten Datenwert-Einträgen,
> Anordnen (640) von Einträgen, die als Spaltennamen-Einträge in der Pivot-Liste gespeichert sind, in entsprechenden Datenwert-Einträgen in entsprechend unterschiedlichen der Wertspalte der Ausgangstabelle, und
> Speichern (660) der Ausgangstabelle direkt in dem Datenbankmanagementsystem, ohne daß sie in ein anderes oder aus einem anderen Format konvertiert wird.

8. Verfahren nach Anspruch 7, **dadurch gekennzeichnet, daß** der Anordnungsschritt für jeden bestimmten Datenwert-Eintrag in jeder bestimmten Spalte der Eingangstabellen-Spalten in der Pivot-Liste ein Anordnen des bestimmten Datenwert-Eintrags in einem solchen Datenwert-Eintrag der Wertspalte der unpivotisierten Ausgangstabelle in einer Zeile umfaßt, die auch einen Datenwert-Eintrag in der Pivot-Spalte entsprechend der bestimmten Spalte der Eingangstabelle enthält.

9. Verfahren nach Anspruch 7, **dadurch gekennzeichnet, daß** die Eingangstabelle mindestens eine andere Spalte (435) als die Spalten in der Pivot-Liste aufweist.

10. Verfahren nach Anspruch 9, **dadurch gekennzeichnet, daß** ein Gruppierungsschritt die Zeilen der Ausgangstabelle im Hinblick auf gleiche Werte der Datenwert-Einträge in der genannten mindestens einen anderen Spalte gruppiert.

11. Relationales Datenbanksystem (200) mit einer Anzahl an Clients (210) und einer Suchmaschine (240) mit Modulen zum Parsen (310), Optimieren (330) und Ausführen (350) einer Anfrage (420, 450) von einem der Clients, die eine Pivotisierungsoperation enthält, die angibt:

eine relationale Eingangstabelle (410) mit Zeilen und Spalten von Datenwerten und mit Spaltennamen, die getrennt von den Datenwerten in den Spalten gespeichert sind,
einen Namen einer Pivot-Spalte (411) in der Eingangstabelle, und
einen Namen einer Wertspalte (412) in der Eingangstabelle,

dadurch gekennzeichnet, daß die Suchmaschine Datenwerte in der Wertspalte um die Pivot-Spalte transponiert, wobei sie mindestens einige der Datenwerte in getrennt gespeicherte und formatierte Spaltennamen umwandelt, so daß sie eine pivotisierte Ausgangstabelle direkt aus der Eingangstabelle aufbaut, ohne eine der beiden Tabellen in ein anderes oder aus einem anderen Format zu konvertieren.

12. System nach Anspruch 11, wobei die Pivotisierungsoperation außerdem eine Pivot-Liste (522) von Datenwerten in der Pivot-Spalte angibt,
dadurch gekennzeichnet, daß die Datenwerte in der Pivot-Spalte aufgrund der Dateneinträge in der Pivot-Liste transponiert werden.

13. System nach Anspruch 12, wobei die Eingangstabelle andere Spalten (413) als die Pivot- und die Wertspalte aufweist,
dadurch gekennzeichnet, daß die Suchmaschine Zeilen der Ausgangstabelle im Hinblick auf gleiche Werte der Datenwert-Einträge in den anderen Spalten gruppiert (550).

14. System nach Anspruch 12, wobei eine Anfrage (440, 460) von einem der Clients eine Entpivotisierungsoperation umfaßt, die angibt:

eine pivotisierte relationale Eingangstabelle (430) mit Zeilen und Spalten von Datenwerten und mit Spaltennamen, die getrennt von den Datenwerten in den Spalten gespeichert sind,
einen Namen (411) für eine Wertspalte,
einen Namen (412) für eine Pivot-Spalte, und
eine Pivot-Liste mit Namen von mindestens einigen der Spalten (431 bis 434) in der pivotisierten Eingangstabelle,

dadurch gekennzeichnet, daß die Suchmaschine die Spaltennamen in der Pivot-Liste um die Pivot-Spalte transponiert (600), wobei sie diese Spaltennamen in getrennt gespeicherte und formatierte Datenwerte umwandelt, so daß sie eine unpivotisierte Ausgangstabelle direkt aus der Eingangstabelle aufbaut, ohne eine von beiden Tabellen in ein anderes oder aus einem anderen Format zu konvertieren.

15. Relationales Datenbanksystem (200) mit einer Anzahl an Clients (210) und einer Suchmaschine (240) mit Modulen zum Parsen (310), Optimieren (330) und Ausführen (350) einer Anfrage (440, 460) von einem der Clients, die eine Entpivotisierungsoperation enthält, die angibt:

eine relationale Eingangstabelle (430) mit Zeilen und Spalten von Datenwerten und mit Spaltennamen, die getrennt von den Datenwerten in den Spalten gespeichert sind,
einen Namen für eine Pivot-Spalte (412), und
eine Pivot-Liste mit Namen von mindestens einigen der Spalten (431 bis 434) in der Eingangstabelle,

dadurch gekennzeichnet, daß die Suchmaschine die Spaltennamen in der Pivot-Liste um die Pivot-Spalte transponiert (600), wobei sie diese Namen in getrennt gespeicherte und formatierte Datenwerte umwandelt, so daß sie eine pivotisierte Ausgangstabelle direkt aus der Eingangstabelle aufbaut, ohne eine von beiden Tabellen in ein anderes oder aus einem anderen Format zu konvertieren.

16. System nach Anspruch 15, wobei die Eingangstabelle andere Spalten (435) als die Pivot- und die Wertspalte aufweist,
dadurch gekennzeichnet, daß die Suchmaschine Zeilen der Ausgangstabelle im Hinblick auf gleiche Werte der Datenwert-Einträge in den anderen Spalten gruppiert.

17. Medium (133) mit Darstellungen von Anweisungen, um einen geeignet programmierten Computer zu veranlassen, ein Verfahren (500) zum Transformieren von Daten aus einer unpivotisierten Eingangstabelle (410) einer relationalen Datenbank in eine pivotisierte Ausgangstabelle (430) auszuführen,
dadurch gekennzeichnet, daß alle folgenden Schritte vollständig innerhalb eines relationalen Datenbankmanagementsystems (200) ausgeführt werden, um Tabellen zu verarbeiten, die Zeilen und Spalten von Einträgen mit Datenwert-Einträ-

gen sowie Einträge mit Spaltennamen, die getrennt von den und in einem anderen Format als die Datenwert-Einträge gespeichert werden, aufweisen, ohne daß eine Konvertierung in ein anderes oder aus einem anderen Gesamttabellenformat stattfindet, mit den folgenden Schritten:

Auswählen (521) des getrennt gespeicherten Namens einer ersten der Spalten der Eingangstabelle als Pivot-Spalte,
Auswählen (523) des getrennt gespeicherten Namens einer zweiten der Spalten der Eingangstabelle als Wertspalte,
Umwandeln (531) eines Satzes an Einträgen, die als Datenwerte in der Wertspalte gespeichert sind, direkt in Einträge, die als Spaltennamen-Einträge in der Ausgangstabelle, gespeichert sind, die pivotisierte Spalten in der Ausgangstabelle benennen,
Anordnen (540) von Einträgen, die als Datenwerte in der Pivot-Spalte gespeichert sind, in entsprechende Datenwert-Einträge in entsprechend unterschiedlichen der pivotisierten Spalten der Ausgangstabelle, und
Speichern (560) der Ausgangstabelle direkt in dem Datenbankmanagementsystem, ohne sie in ein anderes oder aus einem anderen Format zu konvertieren.

18. Medium nach Anspruch 17 mit Darstellungen von Anweisungen, um einen geeignet programmierten Computer zu veranlassen, ein Verfahren zum Transponieren von Daten aus einer pivotisierten Eingangstabelle einer relationalen Datenbank in eine unpivotisierte Ausgangstabelle in der gleichen relationalen Datenbank zu transformieren,
dadurch gekennzeichnet, daß alle folgenden Schritte vollständig innerhalb eines relationalen Datenbankmanagementsystems ausgeführt werden, um Tabellen zu verarbeiten, die Zeilen und Spalten von Einträgen mit Datenwerten sowie Datenwerte mit Spaltennamen, die getrennt von den und in einem anderen Format als die Datenwerteinträge gespeichert sind, aufweisen, ohne daß eine Umwandlung in ein anderes oder aus einem anderen Gesamttabellenformat stattfindet, wobei die Schritte aufweisen:

Auswählen des getrennt gespeicherten Namens einer ersten der Spalten der Eingangstabelle als Pivot-Spalte,
Auswählen einer Vielzahl der getrennt gespeicherten Namen der Spalten der Eingangstabelle als Pivot-Liste,
in der Ausgangstabelle Erzeugen einer Pivot-Spalte mit einem ausgewählten Namen, der in einem Spaltennamen-Eintrag gespeichert ist, und einer Vielzahl getrennt gespeicherten Datenwert-Einträge,
Umwandeln der Spaltennamen-Einträge in der Pivot-Liste in Datenwert-Einträge in der Pivot-Spalte,
in der Ausgangstabelle Erzeugen einer Wertspalte mit einem ausgewählten Namen, der in einem Spaltennamen-Eintrag gespeichert ist und einer Vielzahl separat gespeicherter Datenwert-Einträge,
Anordnen von Einträgen, die als Spaltennamen-Einträge in der Pivot-Liste gespeichert sind, in entsprechende Datenwert-Einträge in entsprechend unterschiedlichen der Wertspalte der Ausgangstabelle, und
Speichern der Ausgangstabelle direkt in dem Datenbankmanagementsystem, ohne sie in ein anderes oder aus einem anderen Format umzuwandeln.

19. Medium (133) mit Darstellungen von Anweisungen, um einen geeignet programmierten Computer zu veranlassen, ein Verfahren (600) zum Transformieren von Daten aus einer pivotisierten Eingangstabelle (430) einer relationalen Datenbank in eine unpivotisierte Ausgangstabelle (410) umzuwandeln,
dadurch gekennzeichnet, daß alle folgenden Schritte vollständig innerhalb eines relationalen Datenbank-managementsystems (200) ausgeführt werden, um Tabellen zu verarbeiten, die Zeilen und Spalten von Einträgen mit Datenwerten sowie Einträge mit Spaltennamen, die getrennt von den und in einem anderen Format als die Datenwert-Einträge gespeichert sind, aufweisen, ohne daß eine Umwandlung in ein anderes oder aus einem anderen Tabellenformat stattfindet, mit den folgenden Schritten:

Auswählen (623) des getrennt gespeicherten Namens einer ersten der Spalten der Eingangstabelle als Pivot-Spalte,
Auswählen (621) einer Vielzahl der getrennt gespeicherten Namen der Spalten der Eingangstabelle als Pivot-Liste, in der Ausgangstabelle Erzeugen (632) einer Wertspalte mit einem ausgewählten Namen, der in einem Spaltennamen-Eintrag gespeichert ist, und einer Vielzahl getrennt gespeicherter Datenwert-Einträge,
Anordnen (640) von Einträgen, die als Spaltennamen-Einträge in der Pivot-Liste gespeichert sind, in entsprechende Datenwert-Einträge in entsprechenden anderen der Wertspalte der Ausgangstabelle, und
Speichern (660) der Ausgangstabelle direkt in dem Datenbankmanagementsystem, ohne sie in ein anderes oder aus einem anderen Format zu konvertieren.

Revendications

1. Un procédé (500), exécuté par un ordinateur, de transformation des données d'une table de bases de données relationnelles d'entrée non pivotée (410) en une table de sortie pivotée (430),

caractérisé en ce que toutes les étapes suivantes sont exécutées entièrement à l'intérieur d'un système de gestion de bases de données relationnelles (200) permettant le traitement de tables contenant des lignes et des colonnes d'éléments contenant des éléments de valeurs de données, et comportant des éléments contenant des noms de colonnes stockés distinctement et dans un format différent des éléments de valeurs de données, sans conversion vers ou depuis un format de table générale différent,, les étapes exécutées par l'ordinateur comprenant:

- la sélection (521) du nom stocké distinctement d'une première des colonnes de la table d'entrée, en tant que colonne pivot;
- la sélection (523) du nom stocké distinctement d'une deuxième des colonnes de la table d'entrée, en tant que colonne de valeurs;
- la conversion (531) d'un ensemble des éléments stockés en tant que valeurs de données dans la colonne pivot, directement en éléments stockés en tant qu'éléments de noms de colonnes dans la table de sortie, indiquant les colonnes pivotées dans la table de sortie;
- le placement (540) d'éléments stockés en tant que valeurs de données dans la colonne pivot, en éléments de valeurs de données respectifs, dans des colonnes correspondantes différentes des colonnes pivotées de la table de sortie;
- le stockage (560) de la table de sortie directement dans le système de gestion de bases de données, sans sa conversion vers un format différent ou depuis un format différent.

2. Le procédé selon la revendication 1, **caractérisé en ce que** l'étape de placement comprend, pour chaque élément de valeurs de données de la colonne des valeurs située dans la même ligne de la table d'entrée, en tant qu'élément particulier de l'ensemble d'éléments de valeurs de données dans la colonne pivot, le placement de l'élément particulier de valeurs de données dans une certaine des colonnes pivotées de la table de sortie pivotée, cette colonne pivotée portant un nom correspondant à l'élément particulier précité de valeurs de données de la colonne pivot.

3. Le procédé selon la revendication 1, **caractérisé par** une autre étape de sélection (522) de certains éléments de l'ensemble d'éléments de valeurs de données de la colonne pivot, en tant que liste pivot, et dans lequel l'étape de conversion convertit, en éléments de noms de colonnes, uniquement ces éléments de valeurs de données de la liste pivot.

4. Le procédé selon la revendication 3, **caractérisé en ce que** l'étape de conversion ne convertit, dans la table d'entrée, aucune ligne qui présente des éléments de valeurs de données NULLES dans la colonne pivot.

5. Le procédé selon la revendication 1, **caractérisé en ce que** la table d'entrée comporte au moins une colonne (413) autre que les colonnes pivot et de valeurs.

6. Le procédé selon la revendication 5, **caractérisé par** le groupage (532), par valeurs égales des éléments de valeurs de données, des lignes de la table de sortie dans au moins une autre colonne.

7. Un procédé (600) de transformation de données, exécuté par un ordinateur, à partir d'une table de bases de données relationnelles d'entrée pivotée (430) en une table de sortie non pivotée (410),

caractérisé en ce que toutes les étapes suivantes sont exécutées entièrement à l'intérieur d'un système de gestion de bases de données relationnelles (200), permettant le traitement de tables contenant des lignes et des colonnes d'éléments contenant des valeurs de données, et comportant des éléments contenant des noms de colonnes stockés distinctement et dans un format différent des éléments de valeurs de données, sans conversion vers ou depuis un format de table générale différent, les étapes exécutées par l'ordinateur comprenant:

- la sélection (623) du nom stocké distinctement d'une première des colonnes de la table d'entrée, en tant que colonne pivot;
- la sélection (621) d'une pluralité des noms stockés distinctement des colonnes de la table d'entrée, en tant que liste pivot;
- la création (632), dans la table de sortie, d'une colonne de valeurs contenant un nom sélectionné stocké dans un élément de noms de colonnes et une pluralité d'éléments de valeurs de données stockés distinctement;
- le placement (640) d'éléments stockés en tant qu'éléments de noms de colonnes de la liste pivot, en éléments de valeurs de données respectifs, dans des colonnes correspondantes différentes de la colonne de valeurs de la table de sortie;
- le stockage (660) de la table de sortie directement dans le système de gestion de bases de données, sans sa conversion vers un format différent.

ou depuis un format différent.

8. Un procédé selon la revendication 7, **caractérisé en ce que** l'étape de placement comprend, pour chaque élément,de valeurs de données particulier de chaque colonne particulière des colonnes de la table d'entrée de la liste pivot, le placement de l'élément des valeurs de données particulier dans l'élément des valeurs de données de la colonne de valeurs de la table de sortie non pivotée dans une ligne qui contient également un élément de valeurs de données dans la colonne pivot correspondant à la colonne particulière de la table d'entrée.

9. Le procédé selon la revendication 7, **caractérisé en ce que** la table d'entrée comporte au moins une colonne (435) autre que les colonnes de la liste pivot.

10. Le procédé selon la revendication 9, **caractérisé en ce que** l'étape de groupage groupe, par valeurs égales des éléments de valeurs de données, les lignes de la table de sortie dans au moins ladite autre colonne.

11. Un système de bases de données relationnelles (200), contenant la liste d'un certain nombre de clients (210) et un moteur de recherche (240), comprenant des modules pour analyser (310), optimiser (330) et exécuter (350) une requête (420, 450), en provenance de l'un des clients, contenant une opération pivotante spécifiant:

- une table d'entrée relationnelle (410) contenant des lignes et des colonnes de valeurs de données et contenant des noms de colonnes stockés distinctement des valeurs de données dans les colonnes,
- un nom d'une colonne pivot (411) dans la table d'entrée, et
- un nom d'une colonne de valeurs (412) dans la table d'entrée,

caractérisé en ce que le moteur de recherche permute (500) des valeurs de données dans la colonne de valeurs autour de la colonne pivot, convertissant au moins quelques-unes des valeurs de données en noms de colonnes stockés et formatés distinctement, de manière à construire une table de sortie pivotée directement depuis la table d'entrée, sans conversion de l'une des deux tables vers un format différent ou depuis un format différent.

12. Le système selon la revendication 11, dans lequel l'opération pivotante spécifie en outre une liste pivot (522) de valeurs de données dans la colonne pivot, **caractérisé en ce que** les valeurs de données dans la colonne pivot sont permutées sur la base des éléments de données dans la liste pivot.

13. Le système selon la revendication 12, dans lequel la table d'entrée comprend des colonnes (413) autres que la colonnes pivot et la colonne de valeurs, **caractérisé en ce que** les moteurs de recherche (550) groupent, dans les autres colonnes, des lignes de la table de sortie en valeurs égales des éléments de valeur de données.

14. Le système selon la revendication 12, dans lequel une requête (440, 460), en provenance de l'un des clients, contient une opération non pivotante spécifiant:

- une table d'entrée relationnelle pivotée (430) contenant des lignes et des colonnes de valeurs de données et comprenant des noms de colonnes stockés distinctement des valeurs de données dans les colonnes,
- un nom (411) pour une colonne de valeurs,
- un nom (412) pour une colonne pivot, et
- une liste pivot contenant des noms d'au moins quelques-unes des colonnes (431-434) de la table d'entrée pivotée,

caractérisé en ce que le moteur de recherche permute (600) les noms de colonnes dans la liste pivot autour de la colonne pivot, convertissant ces noms de colonnes en valeurs de données stockées et formatées distinctement, de manière à construire une table de sortie non pivotée directement à partir de la table d'entrée, sans conversion d'une des deux tables vers un format différent ou depuis un format différent.

15. Un système de bases de données relationnelles (200) contenant la liste d'un certain nombre de clients (210) et un moteur de recherche (240) comprenant des modules pour analyser (310), optimiser (330) et exécuter (350) une requête (440, 460), en provenance de l'un des clients, contenant une opération non pivotante spécifiant:

- une table d'entrée relationnelle (430) contenant des lignes et des colonnes de valeurs de données et contenant des noms de colonnes stockés distinctement des valeurs de données dans les colonnes,
- un nom pour une colonne pivot (412), et
- une liste pivot contenant des noms d'au moins certaines des colonnes (431-434) dans la table d'entrée,

caractérisé en ce que le moteur de recherche permute (600) les noms de colonnes dans la liste pivot autour de la colonne pivot, convertissant ces noms en valeurs de données stockées et formatées distinctement, de manière à construire une table de sortie pivotée directement à partir de la table d'entrée, sans conversion d'une des deux tables vers un format différent ou depuis un format différent.

16. Le système selon la revendication 15, dans lequel la table d'entrée comprend des colonnes (435) autres que la colonne pivot et la colonne de valeurs, **caractérisé en ce que** les moteurs de recherche groupent, dans les autres colonnes, des lignes de la table de sortie en valeurs égales des éléments de valeur de données.

17. Un moyen (133) portant des représentations d'instructions pour permettre à un ordinateur adéquatement programmé d'exécuter un procédé (500) de transformation de données depuis une table de bases de données relationnelles d'entrée non pivotée (410) en une table de sortie pivotée (430),

caractérisé en ce que toutes les étapes suivantes sont exécutées entièrement à l'intérieur d'un système de gestion de bases de données relationnelles (200) permettant le traitement de tables contenant des lignes et des colonnes d'éléments contenant des éléments de valeurs de données, et comportant des éléments contenant des noms de colonnes stockés distinctement et dans un format différent des éléments de valeurs de données, sans conversion vers ou depuis un format de table générale différent, les étapes comprenant:

- la sélection (521) du nom stocké distinctement d'une première des colonnes de la table d'entrée, en tant que colonne pivot;
- la sélection (523) du nom stocké distinctement d'une deuxième des colonnes de la table d'entrée, en tant que colonne de valeurs;
- la conversion (531) d'un ensemble des éléments stockés en tant que valeurs de données dans la colonne pivot directement en éléments stockés en tant qu'éléments de noms de colonnes dans la table de sortie, indiquant les colonnes pivotées dans la table de sortie;
- le placement (540) d'éléments stockés comme valeurs de données dans la colonne pivot, en tant qu'éléments de valeurs de données respectifs dans des colonnes correspondantes différentes des colonnes pivotées de la table de sortie;
- le stockage (560) de la table de sortie directement dans le système de gestion de bases de données, sans sa conversion vers un format différent ou depuis un format différent.

18. Le moyen selon la revendication 17, portant en outre des représentations d'instructions pour permettre à un ordinateur adéquatement programmé d'exécuter un procédé de transformation de données d'une table de bases de données relationnelles d'entrée pivotée en une table de sortie non pivotée dans le même système de bases de données relationnelles,

caractérisé en ce que toutes les étapes suivantes sont exécutées entièrement à l'intérieur d'un système de gestion de bases de données relationnelles permettant le traitement de tables contenant des lignes et des colonnes d'éléments contenant des valeurs de données, et comportant des éléments contenant des noms de colonnes stockés distinctement et ayant un format différent de celui des éléments de valeurs de données, sans conversion vers ou depuis un format de table général différent t , les étapes comprenant:

- la sélection du nom stocké distinctement d'une première des colonnes de la table d'entrée, en tant que colonne pivot;
- la sélection d'une pluralité de noms stockés distinctement des colonnes de la table d'entrée, en tant que liste pivot;
- la création dans la table de sortie d'une colonne pivot contenant un nom sélectionné stocké dans un élément de noms de colonnes et une pluralité d'éléments de valeurs de données stockés distinctement;
- la conversion des éléments de noms de colonnes dans la liste pivot en éléments de valeurs de données dans la colonne pivot;
- la création dans la table de sortie d'une colonne de valeurs contenant un nom sélectionné stocké dans un élément de noms de colonnes et d'une pluralité d'éléments de valeurs de données stockés distinctement;
- le placement d'éléments stockés en tant que valeurs de données dans la colonne pivot, en éléments de valeurs de données respectifs, dans des colonnes correspondantes différentes des colonnes de la table de sortie
- le stockage de la table de sortie directement dans le système de gestion de bases de données, sans sa conversion vers un format différent ou depuis un format différent.

19. Un moyen (133) portant des représentations d'instructions pour permettre à un ordinateur adéquatement programmé d'exécuter un procédé (600) de transformation de données à partir d'une table de bases de données relationnelles d'entrée pivotée (430) en une table de sortie non pivotée (410),

caractérisé en ce que toutes les étapes suivantes sont exécutées entièrement à l'intérieur d'un système de gestion de bases de données relationnelles (200) permettant le traitement de tables contenant des lignes et des colonnes d'éléments contenant des valeurs de données, et comportant des éléments contenant des noms de colonnes stockés distinctement et ayant un format différent de celui des éléments de valeurs de données,, sans conversion vers ou depuis un format de table générale différent,, les étapes comprenant

- la sélection (623) du nom stocké distinctement d'une première des colonnes de la table d'entrée, en tant que colonne pivot;
- la sélection (621) d'une pluralité des noms stockés distinctement des colonnes de la table d'entrée, en tant que liste pivot;.
- la création (632) dans la table de sortie d'une colonne de valeurs contenant un nom sélectionné stocké dans un élément de noms de colonnes et d'une pluralité d'éléments de valeurs de données stockés distinctement;
- le placement (640) d'éléments stockés en tant qu'éléments de noms de colonnes dans la liste pivot sous la forme d'éléments de valeurs de données respectifs dans des colonnes correspondantes différentes des colonnes de valeurs de la table de sortie;
- le stockage (660) de la table de sortie directement dans le système de gestion de bases de données, sans sa conversion vers ou depuis un format différent.

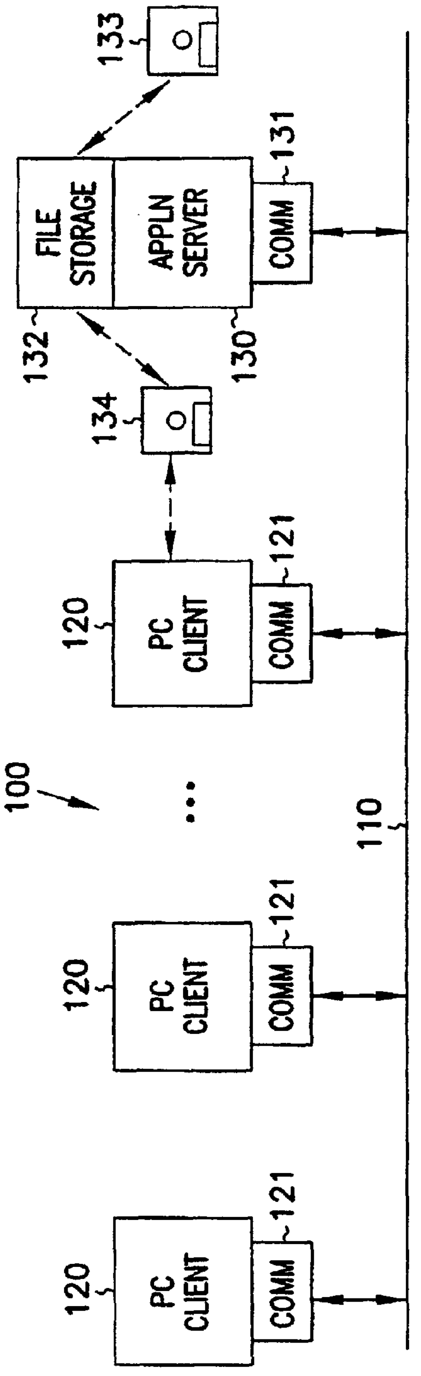
100
PC CLIENT
COMM
120
121
110
FILE STORAGE
APPLN SERVER
132
130
131
133
134

FIG. 1

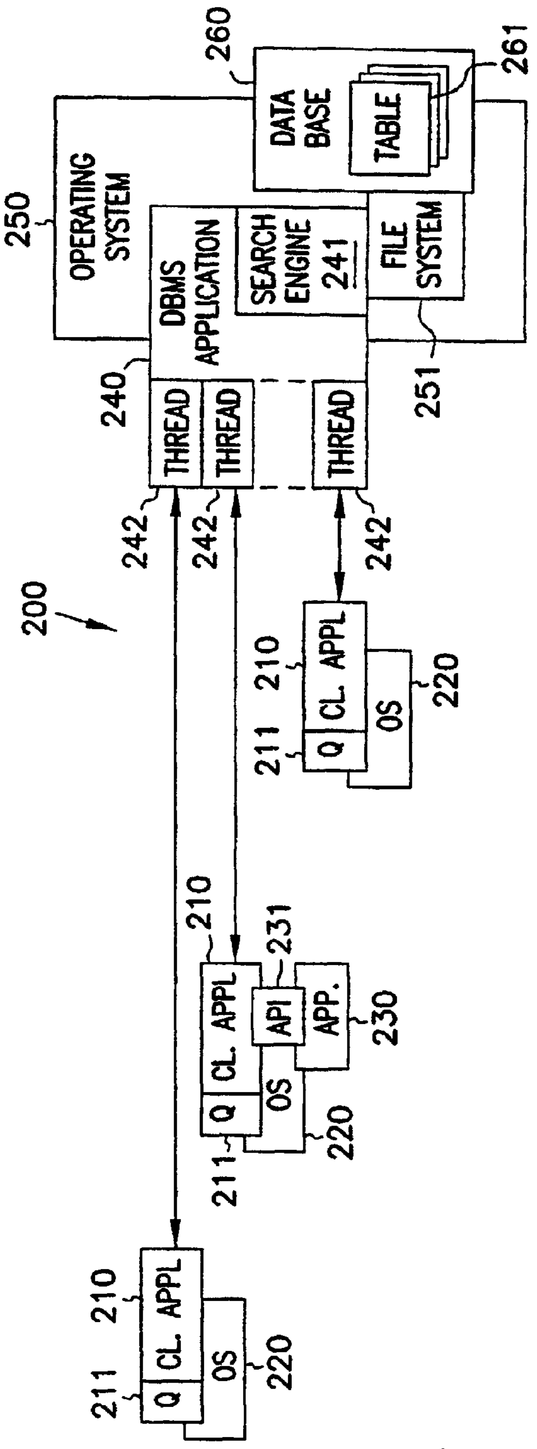
200
Q
CL. APPL
OS
API
APP.
210
211
220
230
231
THREAD
242
240
DBMS APPLICATION
SEARCH ENGINE 241
FILE SYSTEM
251
OPERATING SYSTEM
250
DATA BASE
TABLE
260
261

FIG. 2

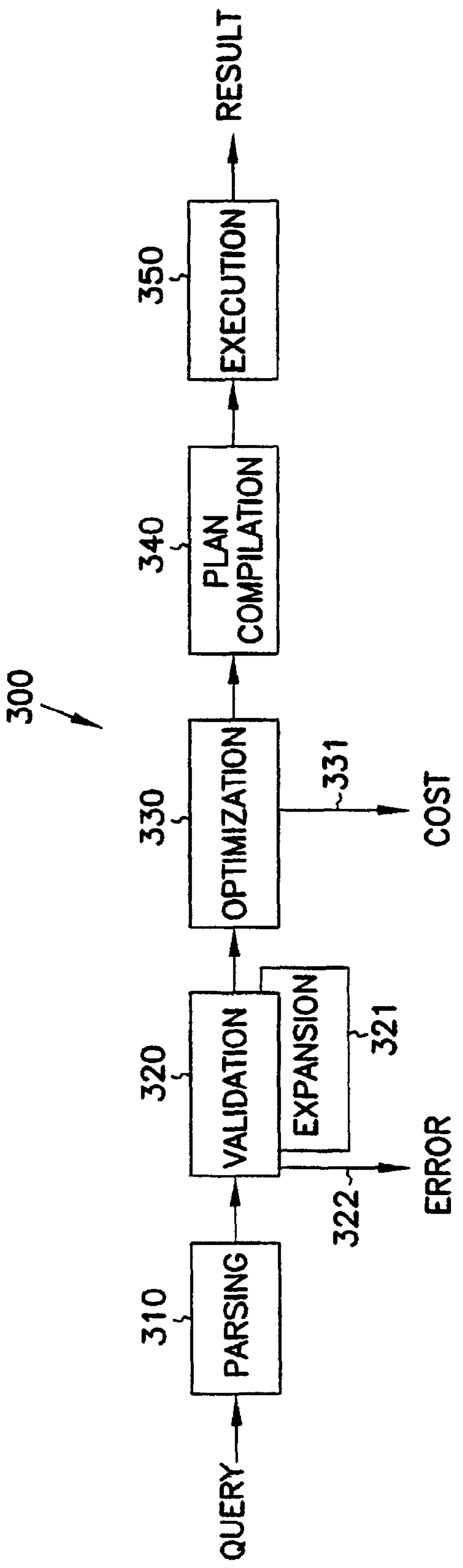
300
QUERY
310
PARSING
320
VALIDATION
321
EXPANSION
322
ERROR
330
OPTIMIZATION
331
COST
340
PLAN COMPILATION
350
EXECUTION
RESULT

FIG. 3

400

NARROW 410

REGION (413)	YEAR (414)	QUARTER (411)	SALES (412)
EAST	1996	SPRING	49000
EAST	1996	SUMMER	51000
EAST	1996	FALL	51000
EAST	1996	WINTER	64000
EAST	1997	SPRING	52000
EAST	1997	SUMMER	54000
EAST	1997	FALL	53000
EAST	1997	WINTER	66000
WEST	1996	SPRING	30000
WEST	1996	SUMMER	31000
WEST	1996	FALL	30000
WEST	1996	WINTER	36000
WEST	1997	SPRING	29000
WEST	1997	SUMMER	29000
WEST	1997	FALL	32000
WEST	1997	WINTER	37000

WIDE 430

REGION (435)	YEAR (436)	SPRING (431)	SUMMER (432)	FALL (433)	WINTER (434)
EAST	1996	49000	51000	51000	64000
EAST	1997	52000	54000	53000	66000
WEST	1996	30000	31000	30000	36000
WEST	1997	29000	29000	32000	37000

NARROW.PIVOT (SALES FOR QUARTER IN (SPRING, SUMMER, FALL, WINTER)) 420

WIDE.UNPIVOT (SALES FOR QUARTER IN (SPRING, SUMMER, FALL, WINTER)) 440

450

NARROW.PIVOT (SALES FOR YEAR IN (1996, 1997) AND QUARTER IN (SPRING, SUMMER, FALL, WINTER))

WIDE.UNPIVOT (SALES OVER (REGION, YEAR)) 460

FIG. 4

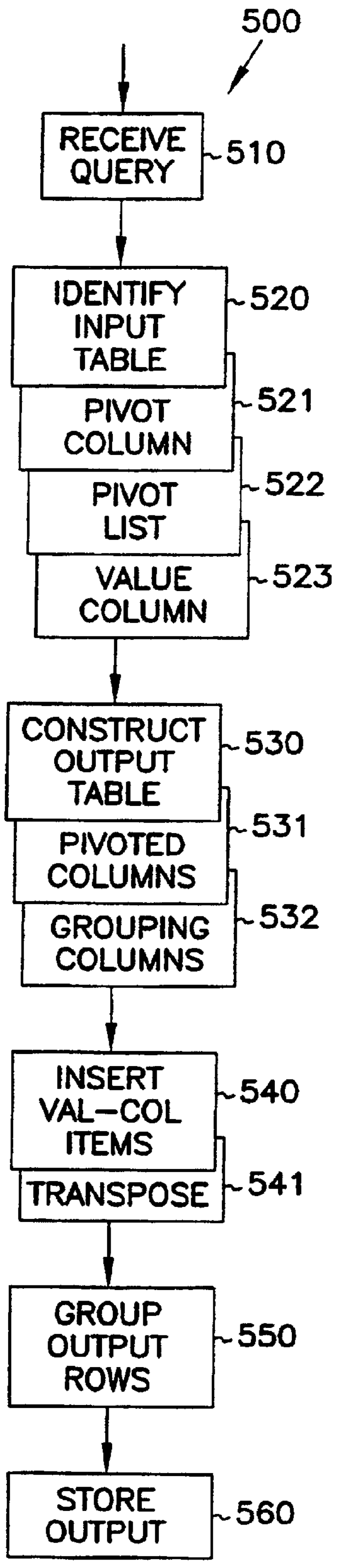
500
RECEIVE QUERY
510
IDENTIFY INPUT TABLE
520
PIVOT COLUMN
521
PIVOT LIST
522
VALUE COLUMN
523
CONSTRUCT OUTPUT TABLE
530
PIVOTED COLUMNS
531
GROUPING COLUMNS
532
INSERT VAL—COL ITEMS
540
TRANSPOSE
541
GROUP OUTPUT ROWS
550
STORE OUTPUT
560

FIG. 5

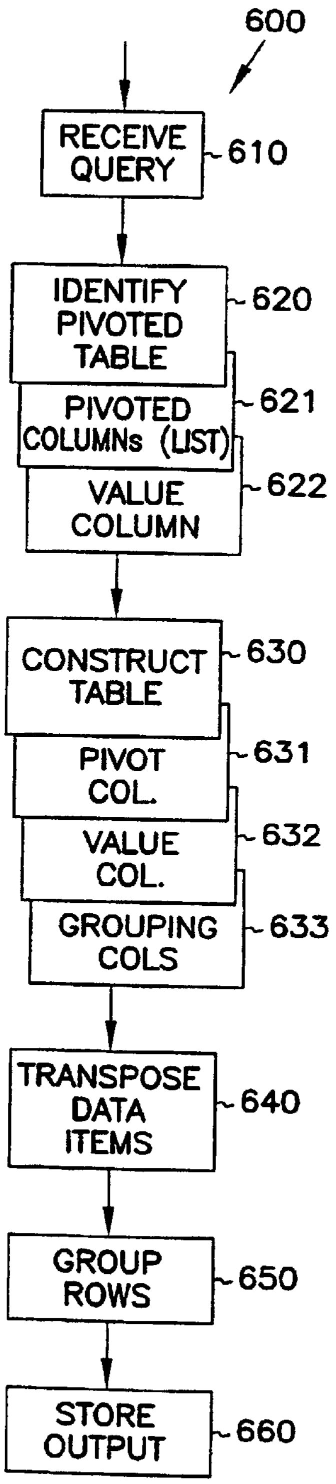
600
RECEIVE QUERY
610
IDENTIFY PIVOTED TABLE
620
PIVOTED COLUMNs (LIST)
621
VALUE COLUMN
622
CONSTRUCT TABLE
630
PIVOT COL.
631
VALUE COL.
632
GROUPING COLS
633
TRANSPOSE DATA ITEMS
640
GROUP ROWS
650
STORE OUTPUT
660

FIG. 6